National University of Computer and Emerging Sciences

Lab Manual 12
Object Oriented Programming

Course Instructor	Syeda Tayyaba Bukhari
Lab Instructor (s)	Fariha Maqbool Muhammad Usama
Section	B
Semester	Spring 2022

Department of Computer Science
FAST-NU, Lahore, Pakistan

Objectives:

- ✓ Exception Handling
- ✓ Template specialization
- ✓ Template Non-type parameters.

Exception Handling:

Exercise 1: Exception Handling Practice

Consider the following C++ code:

```
int numOfItems;
double unitCost;
try
{
cout << "Enter the number of items: ";
cin >> numOfItems;
cout << endl;
if (numOfItems < 0)
throw numOfItems;
cout << "Enter the cost of one item: ";
cin >> unitCost;
cout << endl;
if (unitCost < 0)
throw unitCost;
cout << "Total cost: $"
<< numOfItems * unitCost << endl;
}
catch (int num)
{
cout << "Negative number of items: " << num
<< endl;
cout << "Number of items must be nonnegative."
<< endl;
}
catch (double dec)
{
cout << "Negative unit cost: " << dec
<< endl;
cout << "Unit cost must be nonnegative."
<< endl;
}
```

Answer the following:

a) What is the output if the input is 25 5.50?
b) What is the output if the input is -55 2.8?
c) What is the output if the input is 37 -4.5?
d) What is the output if the input is -10 -2.5?

Exercise 2: Exception Handling Problem

Write a program that prompts the user to enter a person's date of birth in numeric form such as 8-27-1980(Hint: create a class with three data members). The program then outputs the date of birth in the form: August 27, 1980. Your program must contain three exception classes: **invalidDay, invalidMonth,** and **invalidYear**. If the user enters an invalid value for day, then the program should throw and catch an **invalidDay** object. Follow similar convention for the invalid values of month. Handle leap year value with **invalidYear** exception.

Templates

Template Specialization

If we want to define a different implementation for a template when a specific type is passed as template parameter, we can declare a specialization of that template. For example, let's suppose that we have a very simple class called Container that can store one element of any type and that it has just one member function called increase that increases its value by 5 and also returns the increased value. But we find that when it stores an element of type char it would be more convenient to have a completely different implementation of the Container class with a function member uppercase that changes the case of the stored character to the upper case and returns the uppercase character, so we decide to declare a class template specialized for that type.

The general class template looks like:

```
template <class T>
class Container
{
T data;
…
};
```

and the special class template (for char type data) is declared separately as:

```
template<>
class Container<char>
{
char data;
…
};
```

Exercise 3:

Now do the following,

a. Complete the declaration and implementation of these templates with member function increase in the first template and uppercase in the second with the required functionality as stated above. (Note that you are not allowed to use the toupper**()** function) also you are not allowed to create inline functions.
b. Add the following main to your program and watch it run.

```
int main ()
{
Container<int> myint (7);
Container<char> mychar ('j');
cout << myint.increase() << endl;
```

```
cout << mychar.uppercase() << endl;
return 0;
}
```

Practice Question (Not Included in Lab):

Q1.

Non-Type Parameters for Templates

Besides the template arguments that are preceded by the class keyword, which represent types, templates can also have regular typed parameters, similar to those found in functions.

To try this out consider the following class template:

```
template <class T, int N>
class Sequence {
T memblock [N];
public:

void setmember (int x, T value);
T getmember (int x);
};
```

Sequence is a class that stores a Sequence of elements, but here N is an integer. The member function ***setmember*** sets the member at position x in the ***memblock*** with value and ***getmember*** returns the value at index x.

You are required to do the following:

a. Implement the Sequence class. (do not use inline definitions)
b. Copy the following **main** and add it to your program, again watch it run.

```
int main ()
{
Sequence <int,5> myints;
Sequence <double,5> myfloats;
myints.setmember (0,100);
myfloats.setmember (3,3.1416);
cout << myints.getmember(0) << '\n';
cout << myfloats.getmember(3) << '\n';
return 0;
}
```

Q2:

The specifications for the `Pair` class are given below:

- The class **`Pair`** has a private data member `values` which is an array of size two and its type is T(identifier/class/template).
- A constructor that takes two parameters.
- A public member function called `GetMax` that returns the greater of the two stored variables. (This function has to be defined inline i.e. inside the class body).

- A public member function called `GetMin` that returns the smaller of the two stored variables. (This function has to be defined outside the class body). This is done by using the following syntax.

```
template <class identifier>
identifier Pair< identifier >::GetMax(){. . .}
```

- Now replace (which means you have to comment the previous code) the `main` method with the following, the program should run like a river.

```
int main ()
{
  Pair <double> myobject (1.012, 1.01234);
  cout << myobject.getmax();
  return 0;
}
```

As a last exercise you have to change (augment) your code so that the following **main** runs without errors and gives the expected output. Here Pair refers to the class you created earlier, where each element in the Pair has the same type.

```
int main ()
{
Pair <double> y (2.23,3.45);
Sequence <Pair<double>,5> myPairs;
myPairs.setmember (0,y);
cout << myPairs.getmember(0) << '\n';
}
```

Notice that for this code to work you have to overload << operator for the Pair class template. Also the pair class template will need a default constructor. The friend function in this case is declared in a class template using the following syntax:

```
template <class T>

class Pair{

…

template <class identifier> friend ostream & operator << (ostream& out,const
Pair<identifier>& p);

};
```